ERDC/ITL MP-06-1

Information Technology Laboratory

US Army Corps of Engineers ®
Engineer Research and Development Center

System-Wide Water Resources Program

Mesh-Independent Methods for Agent Movement

Jing-Ru C. Cheng

February 2006

System-Wide Water Resources Program

ERDC/ITL MP-06-1
February 2006

Mesh-Independent Methods for Agent Movement

Jing-Ru C. Cheng

Information Technology Laboratory
U.S. Army Engineer Research and Development Center
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

Final report

Prepared for U.S. Army Corps of Engineers
Washington, DC 20314-1000

ABSTRACT: Efficient and accurate methods are needed to move agents (particles with behavior rules) through their environments. To support such applications, this paper presents a compact software architecture that can be used to interface parallel particle tracking software to computational mesh management systems. The in-element particle tracking framework supported by this software architecture is presented in detail. This framework supports most particle tracking applications. The use of this parallel software architecture is illustrated through the implementation of two differential equation solvers, the forward Euler method and an implicit trapezoidal method, on a distributed, unstructured, computational mesh. A design goal of this software effort has been to interface to software libraries such as Scalable Unstructured Mesh Algorithms and Applications (SUMAA3d) in addition to application codes (e.g., FEMWATER). This goal is achieved through a software architecture that specifies a lightweight functional interface that maintains the full functionality required by particle-mesh methods. The use of this approach in parallel programming environments written in C and Fortran is demonstrated.

Contents

Preface

The tests described and the resulting data presented herein, unless otherwise noted, were obtained from research conducted under the sponsorship of the U.S. Army Engineer Districts Walla Walla and Portland and the System-Wide Water Resources Program (SWWRP), a U.S. Army Corps of Engineers research and development initiative. Dr. Steven L. Ashby is the program manager for SWWRP. This work was also supported in part by a grant of computer time from the Department of Defense High Performance Computing Modernization Program at the U.S. Army Engineer Research and Development Center (ERDC) Major Shared Resource Center (MSRC), Information Technology Laboratory (ITL).

This report was prepared by Dr. Jing-Ru C. Cheng, ERDC MSRC, under the supervision of John E. West, Director, ERDC MSRC; Dr. Deborah F. Dent, Deputy Director, ITL; and Dr. Jeffery P. Holland, Director, ITL. It was previously published as a journal article in *Parallel Algorithms and Applications,* Vol. 19 (2-3) 2004, pp. 145-161.

At the time of publication of this report, Dr. James R. Houston was Director of ERDC, and COL James R. Rowan was Commander and Executive Director.

1 Introduction

Agents might represent plants or animals in ecosystems, vehicles in traffic, people in crowds, or autonomous characters in animation and games. Agent-based simulation has been used effectively in ecology, where it is often called individual based modeling (IBM)[1]. Some individual based models are spatially explicit meaning that individuals are associated with a location in geometrical space. Some spatially explicit individual based models also permit explicit mobility, where individuals can move around their environment. This paper presents a portable software architecture for mesh independent parallel particle tracking algorithms, which is central to agent movement in the spatially explicit individual base models on continuous space.

Particle tracking methods occur in a variety of scientific and engineering applications. Such applications are often large-scale and involve the use of high-performance, parallel computing systems for their solution. For such applications the development of scalable particle tracking algorithms and software that can be effectively integrated into existing parallel programming environments is essential. To address these needs, the research is motivated by two main objectives: (1) the development of scalable parallel in-element particle tracking algorithms to efficiently and accurately solve scientific problems by tracking particle movement on two- and three-dimensional unstructured meshes, and (2) the development of a portable software interface to enable a variety of applications requiring particle tracking methods to speed up the development of high-quality parallel application codes.

The remainder of this paper is organized as follows. Section 2 and Section 3 describe the in-element particle tracking method and the associated parallel implementation, respectively. Section 4 presents the software architecture and an Application Programming Interface (API). In Section 5, the results of interfacing with the SUMAA3d [2] parallel programming environment and the parallel FEMWATER [3] application code are presented. Section 6 summarizes these results and discusses planned future work.

2 The In-Element Particle Tracking Method

The particle tracking methods used in the applications mentioned above can be expressed in an algorithmic and software framework called in-element particle tracking. The advantage of this framework is that only local, or in-element, simulation data are required to compute the trajectories of particles at each time-step. Thus, in a parallel computing environment, assigning particles to computational elements that contain them allows for a significant decrease in required interprocessor communication [4]. In this section, the implementation details required for the in-element method is considered; in the next section, the overall parallel algorithm is presented.

The discussion of the in-element particle tracking algorithms in the following sections is simplified by considering massless particles—that is, particles that exactly follow streamlines determined by the local velocity field. However, extending these algorithms to particles with mass on which forces act is a simple

matter. The time-dependent movement of these massless particles is defined by solving the equation

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{v}(\mathbf{x}(t), t) \quad , \tag{1}$$

where $\mathbf{v}$ is the particle velocity at the particle location $\mathbf{x}(t)$ and at time t. In general, the particle location can be obtained at time $t + \triangle t$ by integrating (1)

$$\mathbf{x}(t + \triangle t) = \mathbf{x}(t) + \int_t^{t+\triangle t} \mathbf{v}(\mathbf{x}(\tau), \tau) d\tau \quad . \tag{2}$$

The integral term in (2) must be evaluated numerically because the velocity data are available only on discrete mesh points in most engineering and scientific applications.

Lane [5] summarized the following fundamental issues in particle tracking according to Buning [6, 7] and Post and van Walsum [8]: (1) physical versus computational space tracking, (2) element search, (3) time-step size selection, and (4) velocity interpolation. The in-element particle tracking technique was developed by Cheng et al. [9] using physical space tracking to avoid accuracy loss from the transformation scheme, element-by-element tracking to circumvent the bottleneck of element searching, and subelement tracking to resolve the time-step size selection problem to increase the resolution of velocity field. Additionally, velocities at both current and previous times may be considered in velocity interpolation depending on which ordinary differential equation (ODE) integration scheme is adopted.

The in-element particle tracking method [9] is an algorithm to track particles element by element through the computational mesh. Algorithm 1 approximates the tracking velocity $\mathbf{V}_{\text{track}}$ by averaging the velocities at the current and next time-step, i.e., $\mathbf{V}(t)$ and $\mathbf{V}(t + \triangle t)$, for forward tracking. This approximation results in a significant computational simplification because the element and the edge/face that the particle will intersect can be determined prior to the destination computation. However, the accuracy resulting from velocity interpolation in time may be severely degraded in the case of unsteady flow. In this case, the velocities would have to be interpolated along the particle trajectory. To improve the accuracy of integration, in Algorithm 2 the particle tracking is performed in subelements that are generated by regularly refining the element as the particle passes through it. Obviously, the accuracy is dependent on the order of interpolation. The refinement of regular elements is an approach to increase the order of interpolation scheme. The subelement tracking algorithm (Algorithm 3) includes the determination of the neighbor subelement for the current tracking, the seeking of the edge/face that the particle will lie on, the computation of the ending location on that edge/face, and the calculation of the time spent in this tracking. This algorithm stops and then restarts at each element boundary. A detailed description of this algorithm can be found in [9].

To improve the accuracy of particle tracking in unsteady flow fields, one cannot determine, *a priori*, $\mathbf{V}_{\text{track}}$. Instead, both $\mathbf{V}(t)$ and $\mathbf{V}(t+\triangle t)$ are considered when the neighbor element/subelement needs to be determined. Other items

computed are the residual time, t_r, the ending location, and velocity of the particle. Appendix B in [4] provides insight for the mathematical formulation of the in-element particle tracking under this circumstance.

Algorithm 1 *The time-marching loop for in-element particle tracking*

Foreach $t \leq t_{max}$ *do*
  Read/Compute $\mathbf{V}(\mathbf{x}, t + \triangle t)$
  $\mathbf{V}_{\text{track}}(\mathbf{x}, t + \triangle t) = \frac{1}{2}\left[\mathbf{V}(\mathbf{x}, t) + \mathbf{V}(\mathbf{x}, t + \triangle t)\right]$
  Track particles using $\mathbf{V}_{\text{track}}(\mathbf{x}, t + \triangle t)$
  Update particle data
  $t \leftarrow t + \triangle t$
Endfor

Algorithm 2 *The in-element particle tracking algorithm*

Let P_0 *be the set of particles*
Set the residual time t_r *to the time-step size* $\triangle t$
$n \leftarrow \|P_0\|$
Foreach $(p \in P_0$ *) do*
  Determine the neighbor element M *that the particle will pass through based on velocity rule with* $\mathbf{V}_{\text{track}}(\mathbf{x}, t + \triangle t)$
  While $(t_r > 0)$ *do*
    Refine the element M *to the prescribed number of subelements*
    Track p *subelement by subelement until time is exhausted or until the element boundary is intersected*
    Compute t_r*, velocity, and identify the possible neighbor element* M *for next tracking*
    if $t_r = 0$ *do*
      Update the location of p *at current time*
    Endif
  Endwhile
Endfor

Algorithm 3 *The subelement tracking algorithm*

Determine the neighbor subelement m *that the particle will pass through based on velocity rule with* $\mathbf{V}_{\text{track}}(\mathbf{x}, t + \triangle t)$
Determine the edge/face ℓ *of the subelement* m *that the particle will intersect*
Compute the location on the edge/face ℓ
Calculate the time spent in this tracking
Return the residual time t_r *and the location of particle* p

Figure 1 demonstrates how the algorithms work. From this figure, one can see that the in-element tracking technique computes a path through elements and subelements. In this example, the particle starts from vertex P, passes through elements 2 and 3, and stops at the point Q in element 4. Seven particle tracking

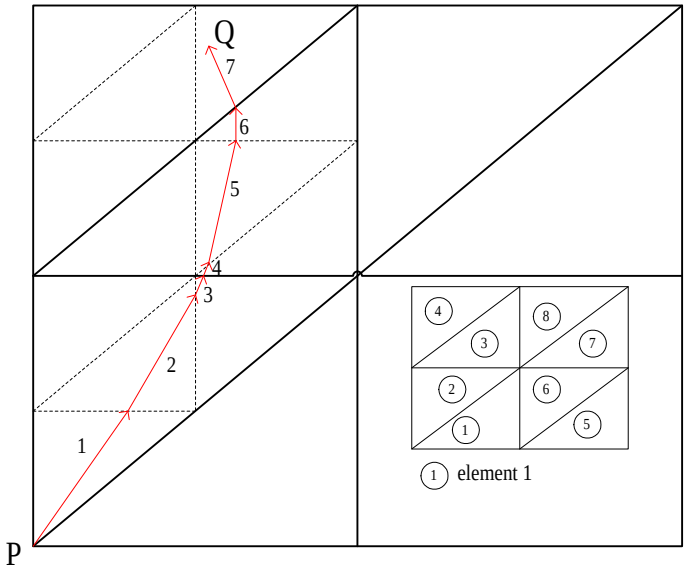

FIGURE 1: An example of in-element particle tracking

paths are shown: three in element 2, three in element 3, and one in element 4. One can observe that elements 1 as well as 5 through 8 are not involved in the tracking process; therefore, refining these elements is not necessary.

Two ODE solvers have been implemented to track particles in subelements: forward Euler and an implicit trapezoidal method. Based on Algorithm 3, the forward Euler and implicit trapezoidal methods are implemented using the approximated tracking velocity $\mathbf{V}_{\text{track}}(\mathbf{x}, t + \triangle t)$. This means a constant velocity $\mathbf{V}_{\text{track}}(\mathbf{x}, t + \triangle t)$ is considered for tracking during the period $\triangle t$. The mathematical formulation (see Appendix B in [4]) has been derived based on the velocities $\mathbf{V}(\mathbf{x}, t)$ and $\mathbf{V}(\mathbf{x}, t + \triangle t)$ to determine the face/edge that the particle will move onto, to compute the location on that face/edge, and to calculate the time spent in this tracking. The following subsections detail the mathematical formulation implemented in the software.

2.1 The forward Euler method (Method 1)

The procedure to determine the neighbor subelement that the particle will pass through can be found in the Appendix B of [4]. Once the subelement has been determined into which the particle will move, the same concept is implemented to determine the edge/face that the particle will intersect.

2.1.1 The determination of the intersection edge/face for an element

Provided first here is a formulation for a two-dimensional (2-D) element. Because the particle p is always on the edge 0-1 after renumbering the vertex number on the element (see Figure 2), a decision function D is defined as

$$D = \hat{k} \cdot (\mathbf{L}_{\text{p2}} \times \mathbf{V}_{\text{track}_\text{p}}) \quad , \tag{3}$$

where the element is assumed to be in the x-y plane and $\hat{k}$ is the unit vector in the z-direction. From Figure 2, one can conclude that (1) edge 0-2 is the target edge if $D > 0$, (2) edge 1-2 is the target edge if $D < 0$, and (3) the particle will move along the line p-2 if $D = 0$.

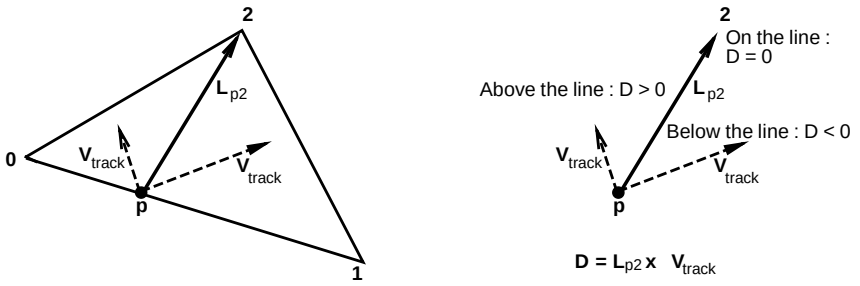

FIGURE 2: In a 2-D triangular subelement, the relationship of $\mathbf{L}_{\mathrm{p2}}$ and $\mathbf{V}_{\mathrm{track}}$ determines the edge/face onto which the particle will move

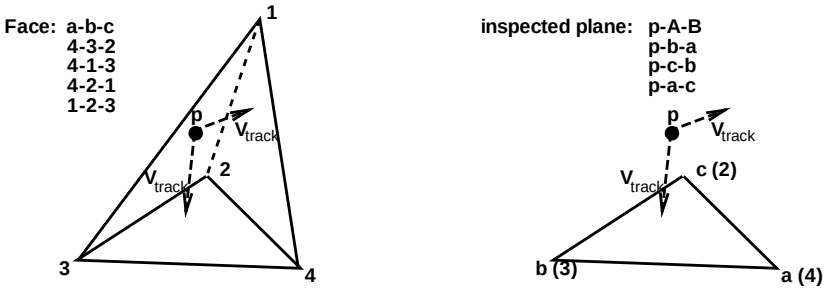

FIGURE 3: Left: In the subelement tracking kernel, the vertex numbers in a tetrahedral element are numbered as the sketch. Right: Three planes are inspected when a face is taken into account

The formulation for the 3-D element is as follows. On the left of Figure 3, a tetrahedral element is used for the demonstration and the definition of four faces (each of them is mapped to a-b-c) as listed in the figure. For example, for the case on the right side of Figure 3, the general face a-b-c would be the specific 4-3-2 face of the element. The following procedure is executed face-by-face to determine the face onto which the particle will move. For the face 4-3-2, three planes (each of them is mapped to p-A-B) are inspected to determine if this is the face for further computation (for particle destination), as depicted in Figure 3. On each plane p-A-B, the quantity D, as shown in Figure 4, is computed as follows.

$$D = \mathbf{V}_{\mathrm{track_p}} \cdot (\mathbf{L}_{\mathrm{pA}} \times \mathbf{L}_{\mathrm{pB}}) \tag{4}$$

From Figure 4, the conclusion is that (1) the particle will move above the plane p-A-B if $D > 0$, (2) the particle will move below the plane if $D < 0$, and (3) the particle will move on the plane if $D = 0$.

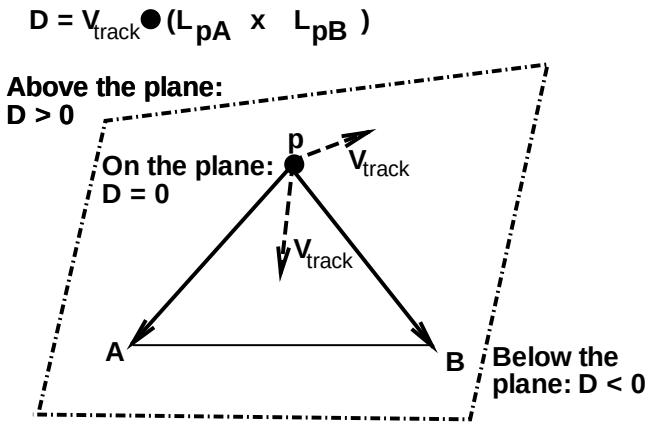

FIGURE 4: The determination of the tracking direction of a particle in a 3-D space

2.1.2 Calculation of the particle destination

For a 2-D particle tracking path, the target edge is a line segment denoted by 1-2 (see Figure 5). The coordinates of vertices 1, 2, and of the particle p are represented by a tuple (x, y) with subscripts 1, 2, and p, respectively. From Figure 5, the interpolation parameter $\xi \in [0, 1]$ is used to locate the particle destination q. Based on the following equation,

$$\frac{\|\mathbf{x}_q - \mathbf{x}_p\|}{\|\mathbf{V}_{\mathrm{track_p}}\|} = \frac{x_q - x_p}{V_x} \tag{5}$$

and the linear interpolation of q written as (7), (6) can then be obtained as a function of ξ.

$$\begin{aligned} f(\xi) &= (\xi(x_1 - x_2) + x_2 - x_p)\,\|\mathbf{V}_{\mathrm{track_p}}\| - V x_{\mathrm{track_p}} \cdot \sqrt{\xi^2 \mathbf{L}_{21}^2 + 2\xi \mathbf{L}_{21} \cdot \mathbf{L}_{\mathrm{p2}} + \mathbf{L}_{\mathrm{p2}}^2} \\ &= 0\ . \end{aligned} \tag{6}$$

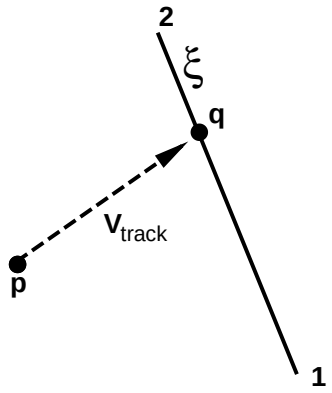

FIGURE 5: The particle moves from p to q, which is on the line segment 1-2

The velocity $\mathbf{V}_{\text{track}}$ can be expressed as $(Vx_{\text{track}}, Vy_{\text{track}})$. Presented now is an approach for solving for the intersection point based on solving a nonlinear set of equations. This approach is not necessary for the forward Euler method (a linear equation can be solved), but this approach can be more easily extended to the trapezoidal method. Once the parameter value ξ is obtained by this method, both the location and velocity at the intersection point q can be calculated. These are given by

$$\begin{aligned} \mathbf{x}_q &= \xi(\mathbf{x}_1 - \mathbf{x}_2) + \mathbf{x}_2 \quad , \; and \\ \mathbf{V}_q &= \xi(\mathbf{V}_1 - \mathbf{V}_2) + \mathbf{V}_2. \end{aligned} \tag{7}$$

The time along this path segment spent is

$$\triangle t = \|\mathbf{L}_{\text{pq}}\| \; / \; \|\mathbf{V}_{\text{track}_\text{p}}\| \; . \tag{8}$$

If the value of $\triangle t$ is larger than the remaining available time, δt, for this tracking, then the location of q, $\mathbf{x}'_q$, is recalculated according to (9),

$$\mathbf{x}'_q = \mathbf{x}_p + (\mathbf{x}_q - \mathbf{x}_p)\frac{\delta t}{\triangle t} \; , \tag{9}$$

and the velocity of q is obtained using the interpolation based on (9).

For a tetrahedral mesh on a 3-D domain, the particle ends on a triangular face. The coordinates of vertices 1, 2, 3, and of the particle are represented by a tuple (x, y, z) with subscripts 1, 2, 3, and p, respectively. On the face, the location and velocity of the target point q are written as the two equations

$$\begin{aligned} \mathbf{x}_q &= N_1 \cdot \mathbf{x}_1 + N_2 \cdot \mathbf{x}_2 + (1 - N_1 - N_2) \cdot \mathbf{x}_3 \\ \mathbf{V}_q &= N_1 \cdot \mathbf{V}_1 + N_2 \cdot \mathbf{V}_2 + (1 - N_1 - N_2) \cdot \mathbf{V}_3 \quad , \end{aligned} \tag{10}$$

where N_1 and N_2 are the two independent variables in the 3-D natural coordinate system and must be within $[-1, 1]$. Similarly as for the 2-D case, the formulation based on

$$\frac{\|\mathbf{x}_q - \mathbf{x}_p\|}{\|\mathbf{V}_{\text{track}_\text{p}}\|} = \frac{x_q - x_p}{V_x} = \frac{y_q - y_p}{V_y} \tag{11}$$

is aimed to extend to Method 2, the trapezoidal method described in the next section. N_1 and N_2 can be determined by using the Newton-Raphson method to solve the following equations obtained from (11) and (10):

$$\begin{aligned} f(N_1, N_2) &= [N_1(x_1 - x_3) + N_2(x_2 - x_3) + x_3 - x_p] \cdot \|\mathbf{V}_{\text{track}_\text{p}}\| - \\ &\quad \|N_1(\mathbf{x}_1 - \mathbf{x}_3) + N_2(\mathbf{x}_2 - \mathbf{x}_3) + \mathbf{x}_3 - \mathbf{x}_p\| \cdot Vx_{\text{track}_\text{p}} \\ &= 0 \quad , \; and \\ g(N_1, N_2) &= [N_1(y_1 - y_3) + N_2(y_2 - y_3) + y_3 - y_p] \cdot \|\mathbf{V}_{\text{track}_\text{p}}\| - \\ &\quad \|N_1(\mathbf{x}_1 - \mathbf{x}_3) + N_2(\mathbf{x}_2 - \mathbf{x}_3) + \mathbf{x}_3 - \mathbf{x}_p\| \cdot Vy_{\text{track}_\text{p}} \\ &= 0 \quad . \end{aligned} \tag{12}$$

Once N_1 and N_2 are obtained, both the location and velocity of q can be calculated with (10). The simulation time elapsed along this particle tracking path is calculated with (8). As with the 2-D case, the location of q is calculated with (9) if the particle finishes tracking at this time-step. The velocity is interpolated based on the new location of q.

2.2 The implicit trapezoidal method (Method 2)

2.2.1 The determination of the intersection edge/face for an element

The determination of the target edge is the same as Method 1 except the definition of D is modified to

$$D = \hat{k} \cdot \mathbf{L}_{\mathrm{p2}} \times \frac{\mathbf{V}_{\mathrm{track_p}} + \mathbf{V}_{\mathrm{track_2}}}{2} \tag{13}$$

for the 2-D mesh, and

$$\begin{aligned} D_1 &= \frac{\mathbf{V}_{\mathrm{track_p}} + \mathbf{V}_{\mathrm{track_A}}}{2} \cdot (\mathbf{L}_{\mathrm{pA}} \times \mathbf{L}_{\mathrm{pB}}) \\ D_2 &= \frac{\mathbf{V}_{\mathrm{track_p}} + \mathbf{V}_{\mathrm{track_B}}}{2} \cdot (\mathbf{L}_{\mathrm{pA}} \times \mathbf{L}_{\mathrm{pB}}) \end{aligned} \tag{14}$$

for the 3-D mesh. To determine whether the particle's tracking direction will be above, on, or below the plane, both D_1 and D_2 are required to be greater than, equal to, or less than zero, respectively.

2.2.2 Calculation of the particle destination

The procedure to derive the equations for calculating particle destination is the same as Method 1 except the equations solved by the Newton-Raphson method for ξ on the 2-D mesh are defined as

$$\begin{aligned} f(\xi) &= \frac{1}{2}\left(\xi(x_1 - x_2) + x_2 - x_p\right) \left\|\xi(\mathbf{V}_{\mathrm{track_1}} - \mathbf{V}_{\mathrm{track_2}}) + \mathbf{V}_{\mathrm{track_p}} + \mathbf{V}_{\mathrm{track_2}}\right\| \\ &\quad -\frac{1}{2}\sqrt{\xi^2\mathbf{L}_{21}^2 + 2\xi\mathbf{L}_{21}\cdot\mathbf{L}_{\mathrm{p2}} + \mathbf{L}_{\mathrm{p2}}^2} \\ &\quad \cdot\left[\xi(Vx_{\mathrm{track_1}} - Vx_{\mathrm{track_2}}) + Vx_{\mathrm{track_p}} + Vx_{\mathrm{track_2}}\right] \\ &= 0 \quad . \end{aligned} \tag{15}$$

The simulation time spent on this path segment is modified from (8) to

$$\triangle t = 2\|\mathbf{L}_{\mathrm{pq}}\| \;/\; \|\xi(\mathbf{V}_{\mathrm{track_1}} - \mathbf{V}_{\mathrm{track_2}}) + \mathbf{V}_{\mathrm{track_p}} + \mathbf{V}_{\mathrm{track_2}}\| \;. \tag{16}$$

For the 3-D case, N_1 and N_2 are obtained by using the Newton-Raphson method to solve the following equations:

$$f(N_1, N_2) = \frac{1}{2}\left[N_1(x_1 - x_3) + N_2(x_2 - x_3) + x_3 - x_p\right] \cdot$$

$$
\begin{aligned}
& \|N_1(\mathbf{V}_{\text{track}_1} - \mathbf{V}_{\text{track}_3}) + N_2(\mathbf{V}_{\text{track}_2} - \mathbf{V}_{\text{track}_3}) + \mathbf{V}_{\text{track}_3} + \mathbf{V}_{\text{track}_\text{p}}\| \\
& -\frac{1}{2}\|N_1(\mathbf{x}_1 - \mathbf{x}_3) + N_2(\mathbf{x}_2 - \mathbf{x}_3) + \mathbf{x}_3 - \mathbf{x}_p\| \cdot [N_1(Vx_{\text{track}_1} - Vx_{\text{track}_3}) \\
& +N_2(Vx_{\text{track}_2} - Vx_{\text{track}_3}) + Vx_{\text{track}_3} + Vx_{\text{track}_\text{p}}\Big] \\
= \;& 0
\end{aligned}
\tag{17}
$$

$$
\begin{aligned}
g(N_1, N_2) \;=\; & \frac{1}{2}\left[N_1(y_1 - y_3) + N_2(y_2 - y_3) + y_3 - y_p\right] \cdot \\
& \|N_1(\mathbf{V}_{\text{track}_1} - \mathbf{V}_{\text{track}_3}) + N_2(\mathbf{V}_{\text{track}_2} - \mathbf{V}_{\text{track}_3}) + \mathbf{V}_{\text{track}_3} + \mathbf{V}_{\text{track}_\text{p}}\| \\
& -\frac{1}{2}\|N_1(\mathbf{x}_1 - \mathbf{x}_3) + N_2(\mathbf{x}_2 - \mathbf{x}_3) + \mathbf{x}_3 - \mathbf{x}_p\| \cdot [N_1(Vy_{\text{track}_1} - Vy_{\text{track}_3}) \\
& +N_2(Vy_{\text{track}_2} - Vy_{\text{track}_3}) + Vy_{\text{track}_3} + Vy_{\text{track}_\text{p}}\Big] \\
= \;& 0 \;\; .
\end{aligned}
$$

The simulation time spent on this path segment is modified from (8) to

$$
\triangle t = 2\|\mathbf{L}_{\text{pq}}\| \;/\; \|N_1(\mathbf{V}_{\text{track}_1} - \mathbf{V}_{\text{track}_3}) + N_2(\mathbf{V}_{\text{track}_2} - \mathbf{V}_{\text{track}_3}) + \mathbf{V}_{\text{track}_3} + \mathbf{V}_{\text{track}_\text{p}}\| \;. \tag{18}
$$

For the remaining situations, the calculation is the same as in Method 1.

3 The Parallel In-Element Framework

From a parallel computing perspective, the advantage of the in-element tracking framework is that only local, or in-element, simulation data are required to compute the trajectories of particles at each time-step. Thus, in a parallel computing environment, assigning particles to the computational elements that contain them requires interprocessor communication [4]. A number of parallel computing issues related to particle tracking applications must be considered. The problem of maintaining a good combined load balancing with element-based computation and particle-based computation is considered in [10]. A mesh-independent interface is considered in [11], and the use of an *a posteriori* error estimator based on particle tracking methods for adaptive mesh refinement is considered in [12].

Because in-element particle tracking methods are implemented with respect to element volumes defined by the computational mesh, a natural approach is to develop a parallel framework based on specifying local, element-based operations. However, to move particles between elements, the algorithms implemented in the parallel particle tracking (PT) software require specific information about mesh partitioning and the coherence of parallel mesh data structures. The PT algorithm presented is based on the correlated partitioning of the particle system and the unstructured element mesh. Particles are partitioned to processors based on their element location—this approach ensures that the data required for the following computation (e.g., the Lagrangian step using the particle tracking methods) involve only data local to a processor. The correspondence between particles and elements is maintained through explicit references in the element

and particle data structures. This correspondence is essential to ensure the correct reassignment of particles between processors. The reassignment occurs when particles move between elements owned by a processor and the ghost elements (elements with shared faces, edges, or vertices that are owned by another processor) [2, 13]. Algorithm 4 is the implementation of the parallel in-element particle tracking technique.

Algorithm 4 *The parallel in-element particle tracking algorithm*

Let P_i *be the set of particles on processor* i
Set the residual time t_r *to the time-step size*
$n \leftarrow \sum_i \|P_i\|$
While $(n > 0)$ *do*
  $n_i \leftarrow 0$
  Foreach $(p \in P_i\ )$ *do*
    Determine the adjacent element M *that the particle will pass through based on velocity rule with* $\mathbf{V}_{\text{track}}(\mathbf{x}, t + \triangle t)$
    While $(p \in P_i$ *and status*$(p) \neq$ *finish) do*
      Refine M *to the prescribed number of subelements*
      Track p *subelement by subelement until time is exhausted or intersects the boundary of the element* M
      Compute t_r*, velocity, and identify the possible neighbor element* M *for next tracking*
      If $t_r > 0$ *do*
        If owner$(M) \neq i$ *do*
          Queue p *and remove* p *from* P_i
          $n_i \leftarrow n_i + 1$
        Endif
      Elseif $t_r = 0$ *do*
        Update locations and values of p *at current time*
        status$(p) \leftarrow$ *finish*
      Endif
    Endwhile
  Endfor
  $n = \sum_i n_i$
  Pack, send and receive messages for each particle queue, then unpack messages to form new P_i
Endwhile
Update values on each vertex

4 A Particle-Mesh API

The data structure associated with each particle must include previous and current locations, user-defined attributes such as a particle's previous and current

velocities/data values, and some additional fields—a pointer to the element that the particle lies within (or will move to), an integer identifying the edge/face number that the particle intersects, and a real number managing the bookkeeping of the residual time remaining while computing the path for the current time-step. The caching of ghost elements ensures that when the target element found by the tracking algorithm is owned by another processor, the particle is correctly moved to that processor. In addition, the assignment of elements is crucial to successful implementation of the algorithm. To ensure correct execution of the parallel algorithm, a consistent numbering of vertices, edges, and faces in each element must be maintained.

A software architecture [4] has been designed to provide support for parallel particle tracking applications on unstructured meshes on high performance parallel computers, clusters, or networks of workstations. This architecture encapsulates the functionality required by most particle tracking applications. A key feature of this architecture is that it inherently allows for parallel implementation, assuming that the underlying mesh software has been parallelized. An API is defined to allow for the easy incorporation of different parallel programming environments. Operations provided by the software include particle object initialization, virtual mesh object initialization, parallel in-element particle tracking, reallocation of particles (i.e., reallocate particles to processors after they move to different elements), particle movement, and vertex data value updates. Portability of the underlying message-passing paradigm is achieved by the use of the MPI message-passing standard [14]. Figure 6 details a section of an application code illustrating the use of the in-element particle tracking method.

In Figure 6, three objects—the particle object `pt_data`, the mesh object `pt_mesh`, and the parallel context `pt_procinfo`—are created by calling the three API functions—`PTinit()`, `PTinit_mesh()`, and `PTinit_procinfo()`, respectively. There are three functions designed for users to call to destroy these three objects. The following six API functions construct the computation of the particle tracking algorithm in the time loop. The API routine `PTrelocate_particles` checks each local particle's resident element to determine whether the particle will be moved and to where. If the tracked particle has been determined not to belong to the current processor, the particle is moved to the nonlocal list from the local list and reassigned to the associated queue, as shown in Figures 7 and 8.

The function `PTgntrak` serves as the first sweep of the parallel in-element particle tracking, which allows for only one element to be traversed. The function `PTnextrak_delay` is called within a `while` loop, which means some number of sweeps calling this function may be included. In this function, each particle tracks its path element-by-element until it enters a ghost element (meaning it needs to be shipped to another processor) or its final destination is determined for this time-step. Therefore, the function `PTrelocate_particles` is called in this routine in order to reassign the particle to different processors. The embedded message-passing routine `PTsend_delay` is called after `PTnextrak_delay`. As described, a nonlocal list has been prepared in `PTrelocate`. The main task of this function is packing the message for each processor, sending the message to each processor, and unpacking the received message for each processor to have a correct local list. When every particle has reached its destination, the function

```
/*** particle object initialization ***/
pt_procinfo = PTinit_procinfo();
pt_data = PTinit();

/*** virtual mesh object initialization ***/
pt_mesh = PTinit_mesh();

while (cur_time < total_time){/* time loop */
  if ( PTgntrak() ){/* 1st sweep tracking */
    PTrelocate_particles();/*move to other proc? If yes, where?*/
  }  /* end if */

  /* following sweeps tracking */
  while (PTnextrak_delay()){
     /* send out the particles and receive the new ones */
     PTsend_delay();
  }  /* end while tracking */

  if (! tracking_only){/*** Lagrangian values ***/
     /* move to other proc? If yes, where*/
     PTcreate_nonlocal_copies();
     /* update vtx values */
     max_value = PTupdate_vtx_u();
  } /* end if vertex value update */
}  /* end while time */

/* clean up */
PTfree_list();
PTfree_mesh();
PTfree_procinfo();
```

FIGURE 6: Code for a simple parallel particle tracking application

`PTcreate_nonlocal_copies` is called before updating the vertex values—this step is required by the Eulerian-Lagrangian methods [15]. This function is designed to ship the particle back to its original processor to be updated because particles (vertices) may have traced their paths to processors different from their original owners. The function `PTupdate_vtx_u` simply updates the values of vertices on the processor that owns them based on the information of the particles. Users must provide enough information in the user data of the particle to fulfill this update.

To incorporate different mesh programming environments efficiently, the PT software uses an abstract particle-mesh interface, specified by an API, to interact with the parallel mesh software programming environment. This interface is illustrated in Figure 9. The PT software architecture contains the particle tracking software implementation, which encapsulates the functionality required by most particle tracking applications. In this way, details of the parallel implementation are hidden from the application programmer. In addition, a particle API is provided to allow users to specify methods to create, retrieve, or modify particle attributes. The gateway between the PT software and the selected mesh programming environment (e.g., SUMAA3d), which may include an API to increase interoperability, is the abstract particle mesh interface (PMI) that has been de-

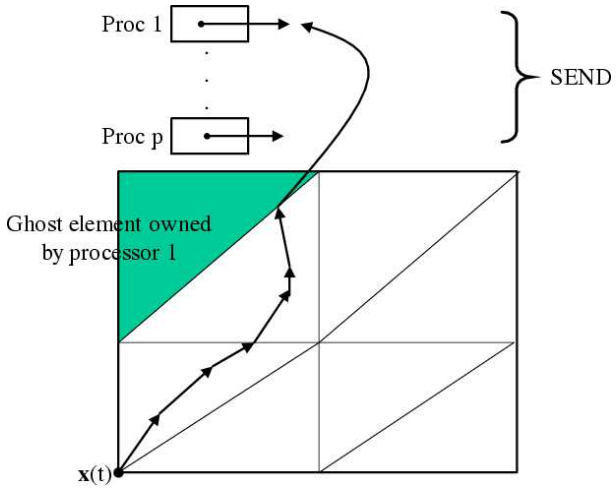

FIGURE 7: An illustration of a the trajectory of a particle requiring it to be reassigned to a different processor

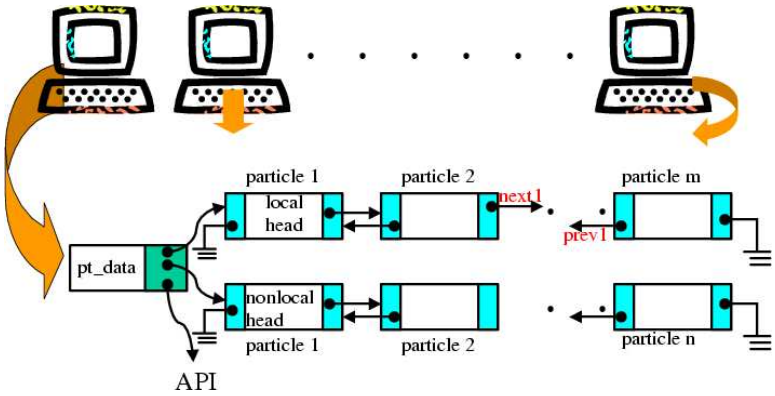

FIGURE 8: The particle object including a `local_head` and a `nonlocal_head` double link lists of particles and a set of methods to access them

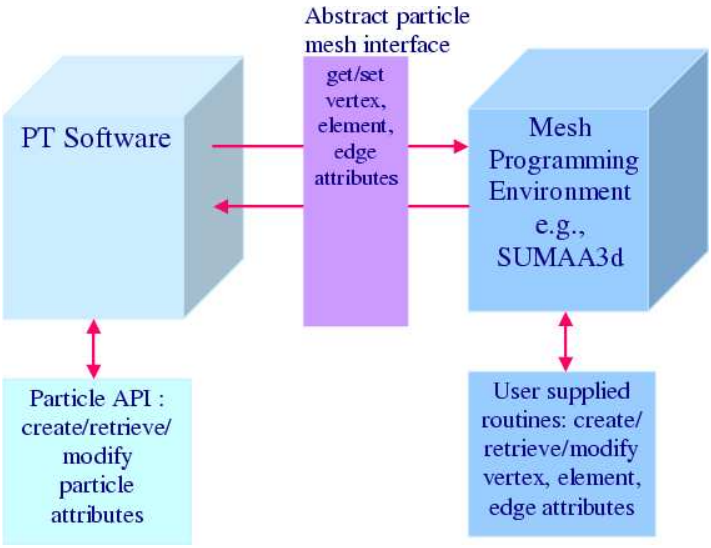

FIGURE 9: The particle tracking software architecture and its interface to an existing mesh programming environment

veloped [4]. The PMI is implemented as a concise functional interface to allow for easy incorporation with different mesh programming environments. This set of functions can be specified by users to provide methods to get/set vertex, element, and edge attributes. Commonly, only a few lines of code are required to implement each function, and this coding effort can often be leveraged by API routines provided by the mesh programming environment. The following is the content of one of the PMI functions named `PMIset_part_list_to_tri()`, which provides a method to set/modify the particle list associated with the element as shown in Figure 10, when SUMAA3d is picked as the mesh programming environment. The API function `RFget_tri_user_data` designed in SUMAA3d is the main access function between the PT software and the SUMAA3d mesh programming environment.

```
void PMIset_part_list_to_tri(PTelm *tri,PTparticle *particle,int w_index)
{ /*@ PMIset_part_list_to_tri - set the particle list to the tri @*/
  tri_data *dptr;
  #ifdef __SOLID3D
    dptr = (tri_data *) RFget_tri_user_data((RFtet *)tri);
  #else
    dptr = (tri_data *) RFget_tri_user_data((RFtriangle *)tri);
  #endif
    dptr->pt_list = particle;
  return;
}
```

5 Experimental Results

In this section, 2-D and 3-D advection-diffusion transport problems are solved using the developed parallel in-element particle tracking method in conjunction with

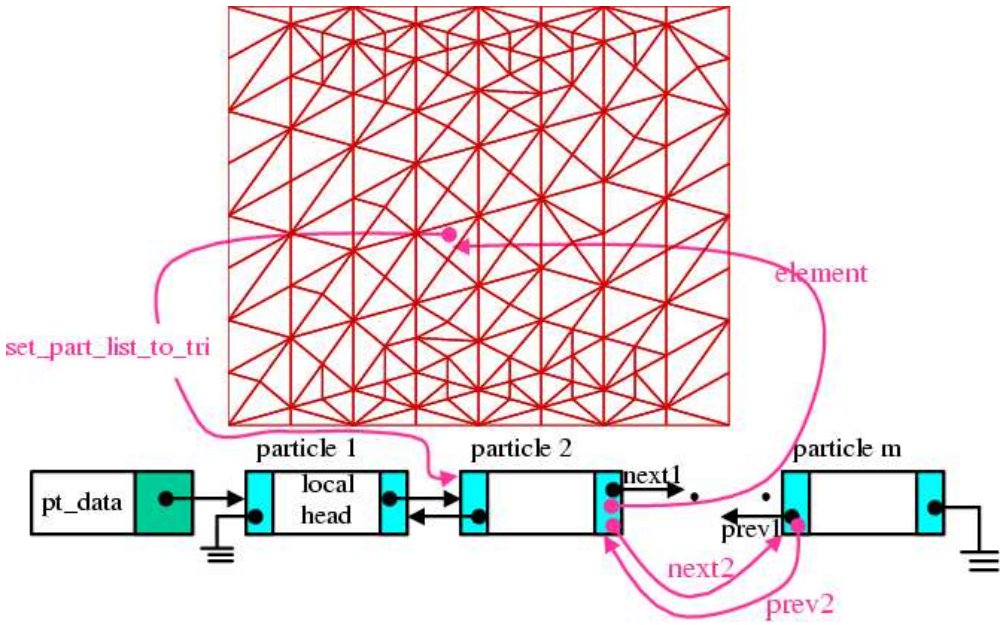

FIGURE 10: The particle-mesh mapping

the SUMAA3d scalable unstructured mesh programming environment. Only a few lines of code are required in each of the particle API and mesh API functions. A 2-D rotating velocity field is adopted to demonstrate the capability of the in-element particle tracking algorithm interfacing with ODE solvers with different orders of accuracy. In addition, the 3-D rotation cone problem, involving pure advection mechanism, is solved to demonstrate the accuracy and performance improved by the parallel implementation.

Test of ODE Solvers

This subsection presents experimental results from testing the parallel in-element approach with two different ODE solvers. The test problem used is derived from the advection of a sharp peaked density cone on a 2-D domain. A 2-D forward particle tracking is performed under the following flow field:

$$\mathbf{u} = (V_x, V_y) = (-\omega y, \omega x) = (\frac{-\pi y}{500}, \frac{\pi x}{500}) \tag{19}$$

where V_x and V_y are the velocity components in the x- and y-directions, respectively. A region of [-3000, 3000]×[-3000,3000] is discretized into triangular elements. The fictitious particles to be forward tracked are originally on vertices in the domain. After a tracking time period of 500, the location of the fictitious particles can be analytically determined by using the relationship in (20) with the time integration from 0 to 500.

$$\mathbf{x} - \mathbf{x}_o = \int_0^{500} \mathbf{u}(\mathbf{x}, t)dt \tag{20}$$

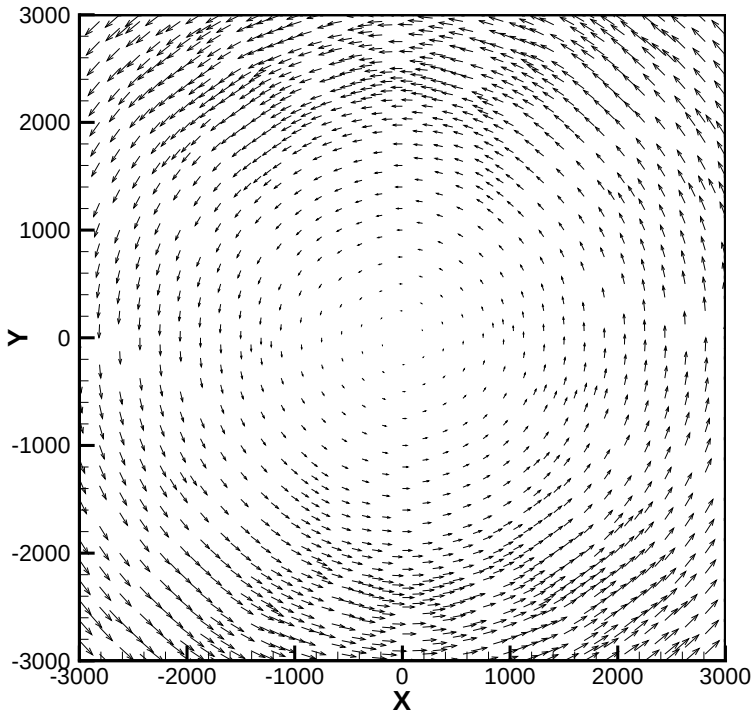

FIGURE 11: Hedgehog plot of 2-D rotating velocity field

Figure 11 demonstrates the counterclockwise velocity field in the domain. In Figure 12, the particle tracking results using two solvers (Method 1 and Method 2 described in Section 2) show that the approximated implicit trapezoidal method (Method 2) matches much better with the analytical solution than the forward Euler method (Method 1) for the four randomly picked streamlines. This result illustrates the correctness of the implementation of the in-element parallel particle tracking algorithm when interfacing with different solvers.

Accuracy and Performance

In this subsection, a 3-D pure advection transport problem is solved using the developed parallel in-element PT software in conjunction with the SUMAA3d scalable unstructured mesh programming environment (written in C) and the parallel FEMWATER program (written in FORTRAN). Only a few lines of coding are required in each of the particle API and mesh API functions. Because these two applications solve the same problem, the particle API functions associated with each of them are defined to access the particle attributes in the same way. However, the PMI functions are built differently to access mesh data created by different programming languages.

For example, in SUMAA3d, the object `RFadaptive_mesh` provides all of the information, including edge neighbors and face neighbors, that is required by the PT software. In parallel FEMWATER, a virtual particle object and mesh object

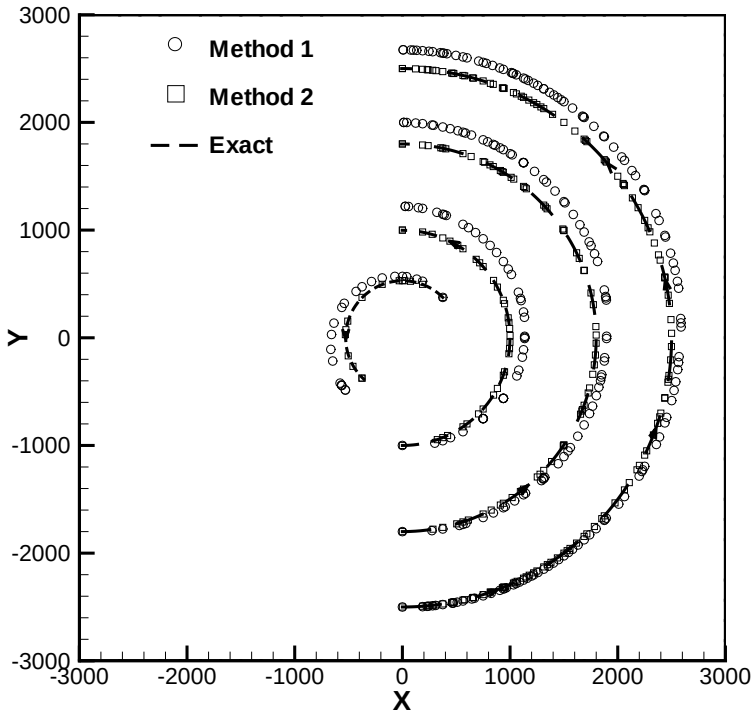

FIGURE 12: Comparison of ODE solutions obtained by Methods 1 and 2

are created in the FORTRAN main program, but they are built in the library function `PTinit` and in the function `PTfw_mesh` to access the FORTRAN-style mesh data structures. Moreover, the user data associated with elements and associated with vertices need to be defined to build an explicit reference between particles and elements and access user data—concentration, the Lagrangian concentration, and three components of velocity—allocated in the FORTRAN code. This extra work is required with the application code FEMWATER and not with the SUMAA3d programming environment. The following is the content of the PMI function `PMIget_part_list_from_tri` when used with SUMAA3d.

```
PTparticle *PMIget_part_list_from_tri(void *mesh, PTelm *tri)
{ /*@ PMIget_part_list_from\_tri - get the particle
      list from the tri. in conjunction with SUMAA3d.
  @*/
  tri_data *dptr;
  #ifdef __SOLID3D
     dptr = (tri_data *) RFget_tri_user_data((RFtet *)tri);
  #else
     dptr = (tri_data *) RFget_tri_user_data((RFtriangle *)tri);
  #endif
  return (PTparticle *)(dptr->pt_list);
}
```

The following code fragment is used with the parallel application code FEMWATER.

```
PTparticle *PMIget_part_list_from_tri(void *mesh,PTelm *tri)
{ /*@ PMIget_part_list_from_tri - get the particle list from
      the tri. in conjunction with parallel FEMWATER.
  @*/
  tri_data *dptr;
  dptr = ((fw_mesh_t *)mesh)->tri_user_data+(
         (int *)tri-((fw_mesh_t *)mesh)->ie) / 9;
  return (PTparticle *)(dptr->pt_list);
}
```

In all, the effort required to construct these PMI functions is relatively small. Maintaining the PT software is much easier than maintaining both SUMAA3d and parallel FEMWATER. The simulation result from the parallel in-element tracking algorithm using the SUMAA3d mesh programming environment is presented in Figure 13. The maximum absolute pointwise error of this simulation is less than 1 percent. The incorporated FEMWATER program was run on the U.S. Army Engineer Research and Development Center Major Shared Resource Center's (ERDC MSRC's) Compaq SC45 system (128 nodes, 512 processors, 1,000 MHz CPUs)—a distributed memory parallel computer based on the Alpha 21264 processor. Figure 14 depicts the wall-clock time spent in particle tracking as a function of the number of processors varying from 2 to 256 using log-log scales. As expected, the slope of the log-log plot in Figure 14 (i.e., speedup in tracking) is close to but less than 1.

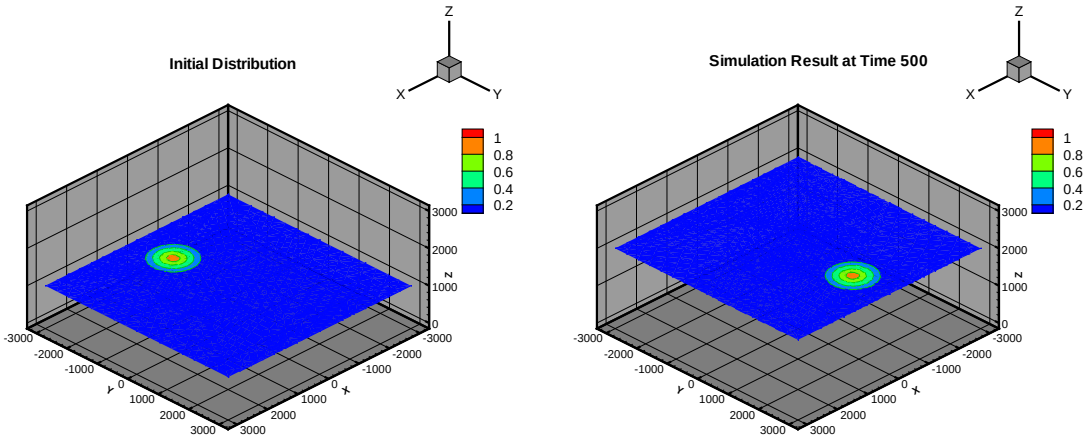

FIGURE 13: Initial distribution (left) and simulation result at time 500 (right) for the 3-D rotation cone problem

6 Conclusion and Future Work

As demonstrated in the preceding section, the building blocks for parallel particle tracking methods in conjunction with different programming environments

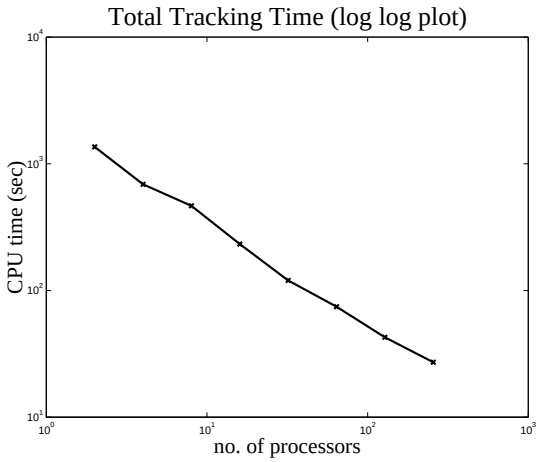

FIGURE 14: The performance of the particle tracking benchmark as a function of the number of processors on the Compaq SC45 system

are short and straightforward. The main reason is that the parallelization is embedded in the PT software, and the PT software provides a carefully designed mesh-particle interface. Application programs, such as ADH [16], pWASH123D [17], ADCIRC [18], pNFS [19], etc., are expected to benefit from the development of the PT software.

The development of agent-based computing framework is an ongoing task to support an environment in which the interactions between agents occur based on their behavior rules. Large-scale agent based simulations are expected to run on a large number of processors on ERDC MSRC machines for performance analysis such as scalability and detailed profiling. In the future, this API may be redefined using more advanced language capabilities, and a parallel particle tracking component is expected to be built to be compliant with the Common Component Architecture Forum (CCA) [20].

Acknowledgments

The tests described and the resulting data presented herein, unless otherwise noted, were obtained from research conducted under the sponsorship of the U.S. Army Districts Walla Walla and Portland and the System-Wide Water Resources Program (SWWRP), a U.S. Army Corps of Engineers research and development initiative. This work was also supported in part by a grant of computer time from the Department of Defense High Performance Computing Modernization Program at the U.S. Army Engineer Research and Development Center Major Shared Resource Center, Information Technology Laboratory, Vicksburg, Mississippi.

References

[1] R. A. Goodwin, J. M. Nestler, J. J. Anderson, L. J. Weber, and D. P. Loucks. Decoding 3-D movement rules of fish for forecasting using a coupled Eulerian-Lagrangian-Agent framework. *Journal of Ecological Modeling*, 2005. in press.

[2] Lori Freitag, Mark Jones, Carl Ollivier-Gooch, and Paul Plassmann. SUMAA3d Web page. http://www.mcs.anl.gov/sumaa3d/, Mathematics and Computer Science Division, Argonne National Laboratory, 1997.

[3] F. Tracy. Parallelization of finite element method flow and transport groundwater computations. In *Proceedings of the Southern Conference on Computing*, The University of Southern Mississippi, MS, Oct. 26–28, 2000.

[4] J-R. C. Cheng. *Parallel Particle Tracking Algorithms and Software for Applications in Scientific Computing.* PhD thesis, Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA, USA, 2002.

[5] David A. Lane. Scientific visualization of large-scale unsteady fluid flows. In *Scientific Visualization Surveys, Methodologies and Techiniques*, pages 125–145, Los Alamitos, CA, 1996. IEEE Computer Society Press.

[6] P. Buning. Numerical algorithms in CFD post-processing, computer graphics and flow visualization in computational fluid dynamics, 1989. von Karman Institute for Fluid Dynamics Lecture Series 1989-07.

[7] P. Bunning. Sources of error in the graphical analysis of CFD results. *Journal of Scientific Computing*, 3(2):149–164, 1988.

[8] F. Post and T. van Walsum. Fluid flow visualization. In G. Nielson H. Hagen, H. Mueller, editor, *Focus on Scientific Visualization*, pages 1–40. Springer, Berlin, 1993.

[9] H.-P. Cheng, J.-R. Cheng, and G. T. Yeh. A particle tracking technique for the Lagrangian-Eulerian finite-element method in multi-dimensions. *International Journal for Numerical Methods in Engineering*, 39(7):1115–1136, 1996.

[10] Jing-Ru C. Cheng and Paul E. Plassmann. A parallel particle tracking framework for applications in scientific computing. *The Journal of Supercomputing*, 28:149–164, 2004.

[11] Jing-Ru C. Cheng and Paul E. Plassmann. A software architecture for parallel particle tracking algorithms. In H.R. Arabnia and Y. Mun, editors, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'03)*, volume II, pages 656–661. CSREA Press, Las Vegas, Nevada, USA, June 23-26, 2003 2003.

[12] Jing-Ru C. Cheng and Paul E. Plassmann. An a posteriori error estimator for adaptive mesh refinement using parallel in-element particle tracking methods. In José M.L.M. Palma, Jack Dongarra, Vicente Hernández,

and A. Augusto Sousa, editors, *High Performance Computing for Computational Science—VECPAR 2002*, volume 2565 of *Lecture Notes in Computer Science*, pages 94–107. Springer-Verlag, 2003.

[13] Mark T. Jones and Paul E. Plassmann. Computational results for parallel unstructured mesh computations. *Computing Systems in Engineering*, 5(4–6):297–309, 1994.

[14] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22:789–828, 1996. MCS-P567-0296.

[15] J.-R. Cheng, H.-P. Cheng, and G. T. Yeh. A Lagrangian-Eulerian method with adaptively local zooming and peak/valley capturing approach to solve two-dimensional advection-diffusion transport equations. *International Journal for Numerical Methods in Engineering*, 39(6):987–1016, 1996.

[16] Kimberlie Staheli, Joseph H. Schmidt, and Spencer Swift. Guidelines for solving groundwater problems with ADH. manuscript, January 1998.

[17] J.-R. C. Cheng, R. M. Hunter, H.-P. Cheng, and D. R. Richards. A parallel software development for watershed simulations. In *Computational Science — ICCS 2005, The Springer Verlag Lecture Notes in Computer Science (LNCS 3514) Series*, pages 460–468. Springer, 2005. Best Paper Award.

[18] ADCIRC Development Group. Advanced Circulation Model (ADCIRC) Web page. http://www.nd.edu/∼adcirc/.

[19] J.-R. C. Cheng, R. M. Hunter, P. McAllister, and D. R. Richards. A software framework for parallel agent-based applications. In *SIAM Conf. on Parallel Processing for Scientific Computing*, San Francisco, CA, Feb. 22-24, 2006.

[20] DOE Laboratories and DOE-supported Universities. CCA Forum. http://www.acl.lanl.gov/cca/.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
February 2006	Final report	

4. TITLE AND SUBTITLE

Mesh-Independent Methods for Agent Movement

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Jing-Ru C. Cheng

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

U.S. Army Engineer Research and Development Center
Information Technology Laboratory
3909 Halls Ferry Road
Vicksburg, MS 39180-6199

8. PERFORMING ORGANIZATION REPORT NUMBER

ERDC/ITL MP-06-1

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Corps of Engineers
Washington, DC 20314-1000

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Efficient and accurate methods are needed to move agents (particles with behavior rules) through their environments. To support such applications, this paper presents a compact software architecture that can be used to interface parallel particle tracking software to computational mesh management systems. The in-element particle tracking framework supported by this software architecture is presented in detail. This framework supports most particle tracking applications. The use of this parallel software architecture is illustrated through the implementation of two differential equation solvers, the forward Euler method and an implicit trapezoidal method, on a distributed, unstructured, computational mesh. A design goal of this software effort has been to interface to software libraries such as Scalable Unstructured Mesh Algorithms and Applications (SUMAA3d) in addition to application codes (e.g., FEMWATER). This goal is achieved through a software architecture that specifies a lightweight functional interface that maintains the full functionality required by particle-mesh methods. The use of this approach in parallel programming environments written in C and Fortran is demonstrated.

15. SUBJECT TERMS

Agent-based simulation
Application programming interface
Individual-based simulation
Parallel computing
Particle tracking
Scientific computing
Software Architecture
SWWRP
System-Wide Water Resources Program

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			19b. TELEPHONE NUMBER *(include area code)*
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED		26	